20 | 详解时钟轮在RPC中的应用

2020-04-06 何小锋

RPC实战与核心原理 进入课程 >

讲述：张浩

时长 11:07 大小 10.19M

你好，我是何小锋。上一讲我们学习了在分布式环境下如何快速定位问题，简单回顾下重点。在分布式环境下，RPC 框架自身以及服务提供方的业务逻辑实现，都应该对异常进行合理地封装，让使用方可以根据异常快速地定位问题；而在依赖关系复杂且涉及多个部门合作的分布式系统中，我们也可以借助分布式链路跟踪系统，快速定位问题。

现在，切换到咱们今天的主题，一起看看时钟轮在 RPC 中的应用。

定时任务带来了什么问题？

在讲解时钟轮之前，我们先来聊聊定时任务。相信你在开发的过程中，很多场景都会使用到定时任务，在 RPC 框架中也有很多地方会使用到它。就以调用端请求超时的处理逻辑为

例，下面我们看一下 RPC 框架是如果处理超时请求的。

回顾下[第 17 讲]，我讲解 Future 的时候说过：无论是同步调用还是异步调用，调用端内部实行的都是异步，而调用端在向服务端发送消息之前会创建一个 Future，并存储这个消息标识与这个 Future 的映射，当服务端收到消息并且处理完毕后向调用端发送响应消息，调用端在接收到消息后会根据消息的唯一标识找到这个 Future，并将结果注入给这个 Future。

那在这个过程中，如果服务端没有及时响应消息给调用端呢？调用端该如何处理超时的请求？

没错，就是可以利用定时任务。每次创建一个 Future，我们都记录这个 Future 的创建时间与这个 Future 的超时时间，并且有一个定时任务进行检测，当这个 Future 到达超时时间并且没有被处理时，我们就对这个 Future 执行超时逻辑。

那定时任务该如何实现呢？

有种实现方式是这样的，也是最简单的一种。每创建一个 Future 我们都启动一个线程，之后 sleep，到达超时时间就触发请求超时的处理逻辑。

这种方式吧，确实简单，在某些场景下也是可以使用的，但弊端也是显而易见的。就像刚才我讲的那个 Future 超时处理的例子，如果我们面临的是高并发的请求，单机每秒发送数万次请求，请求超时时间设置的是 5 秒，那我们要创建多少个线程用来执行超时任务呢？超过 10 万个线程，这个数字真的够吓人了。

别急，我们还有另一种实现方式。我们可以用一个线程来处理所有的定时任务，还以刚才那个 Future 超时处理的例子为例。假设我们要启动一个线程，这个线程每隔 100 毫秒会扫描一遍所有的处理 Future 超时的任务，当发现一个 Future 超时了，我们就执行这个任务，对这个 Future 执行超时逻辑。

这种方式我们用得最多，它也解决了第一种方式线程过多的问题，但其实它也有明显的弊端。

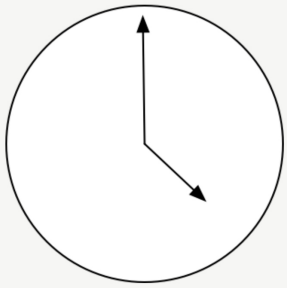
同样是高并发的请求，那么扫描任务的线程每隔 100 毫秒要扫描多少个定时任务呢？如果调用端刚好在 1 秒内发送了 1 万次请求，这 1 万次请求要在 5 秒后才会超时，那么那个扫描的线程在这个 5 秒内就会不停地对这 1 万个任务进行扫描遍历，要额外扫描 40 多次（每 100 毫秒扫描一次，5 秒内要扫描近 50 次），很浪费 CPU。

在我们使用定时任务时，它所带来的问题，就是让 CPU 做了很多额外的轮询遍历操作，浪费了 CPU，这种现象在定时任务非常多的情况下，尤其明显。

什么是时钟轮?

这个问题也不难解决，我们只要找到一种方式，减少额外的扫描操作就行了。比如我的一批定时任务是 5 秒之后执行，我在 4.9 秒之后才开始扫描这批定时任务，这样就大大地节省了 CPU。这时我们就可以利用时钟轮的机制了。

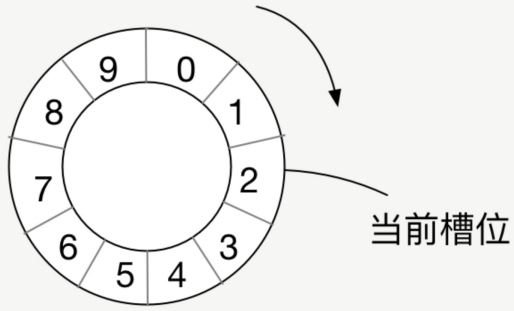
我们先来看下我们生活中用到的时钟。



时钟示意图

很熟悉了吧，时钟有时针、分针和秒针，秒针跳动一周之后，也就是跳动 60 个刻度之后，分针跳动 1 次，分针跳动 60 个刻度，时针走动一步。

而时钟轮的实现原理就是参考了生活中的时钟跳动的原理。



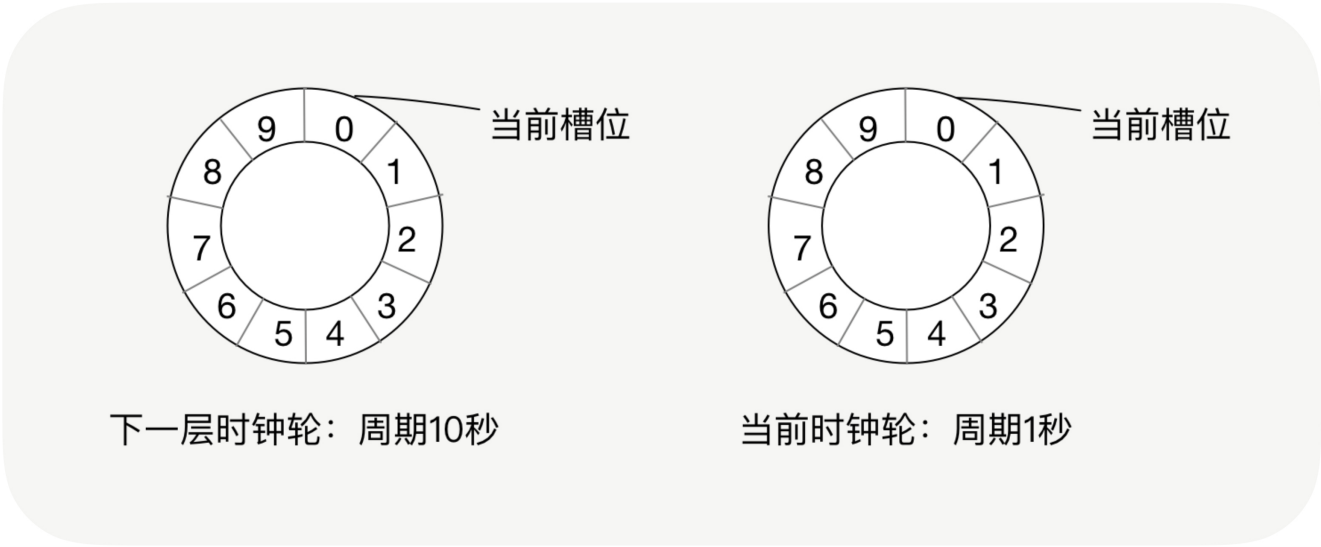
时钟轮示意图

在时钟轮机制中，有时间槽和时钟轮的概念，时间槽就相当于时钟的刻度，而时钟轮就相当于秒针与分针等跳动的一个周期，我们会将每个任务放到对应的时间槽位上。

时钟轮的运行机制和生活中的时钟也是一样的，每隔固定的单位时间，就会从一个时间槽位跳到下一个时间槽位，这就相当于我们的秒针跳动了一次；时钟轮可以分为多层，下一层时钟轮中每个槽位的单位时间是当前时间轮整个周期的时间，这就相当于 1 分钟等于 60 秒钟；当时钟轮将一个周期的所有槽位都跳动完之后，就会从下一层时钟轮中取出一个槽位的任务，重新分布到当前的时钟轮中，当前时钟轮则从第 0 槽位从新开始跳动，这就相当于下一分钟的第 1 秒。

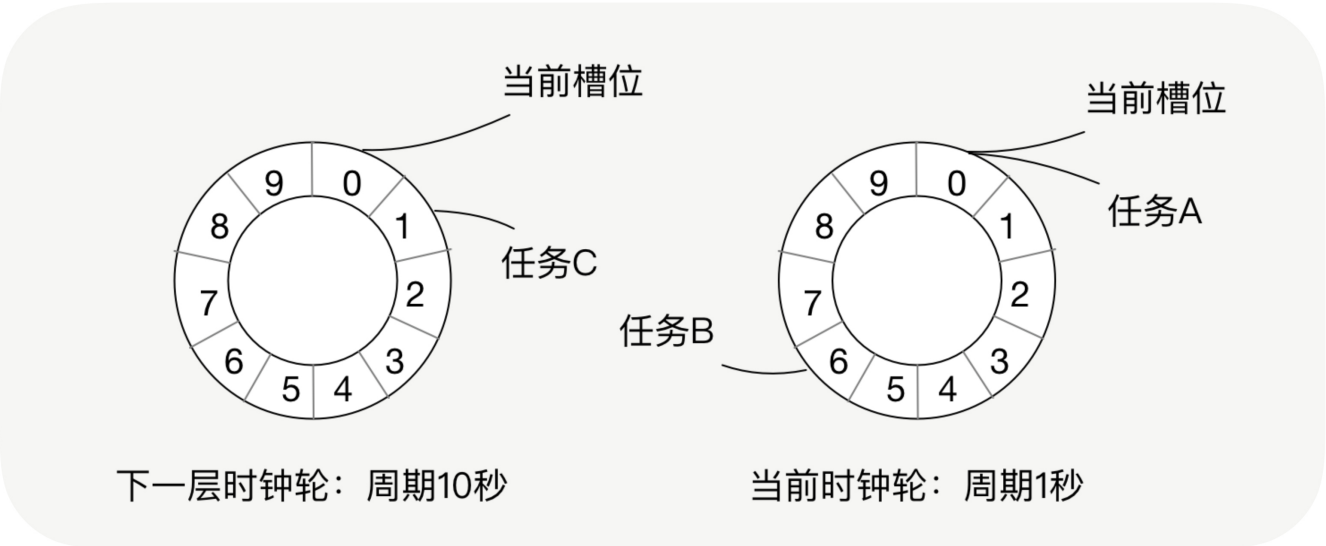
为了方便你了解时钟轮的运行机制，我们用一个场景例子来模拟下，一起看下这个场景。

假设我们的时钟轮有 10 个槽位，而时钟轮一轮的周期是 1 秒，那么我们每个槽位的单位时间就是 100 毫秒，而下一层时间轮的周期就是 10 秒，每个槽位的单位时间也就是 1 秒，并且当前的时钟轮刚初始化完成，也就是第 0 跳，当前在第 0 个槽位。



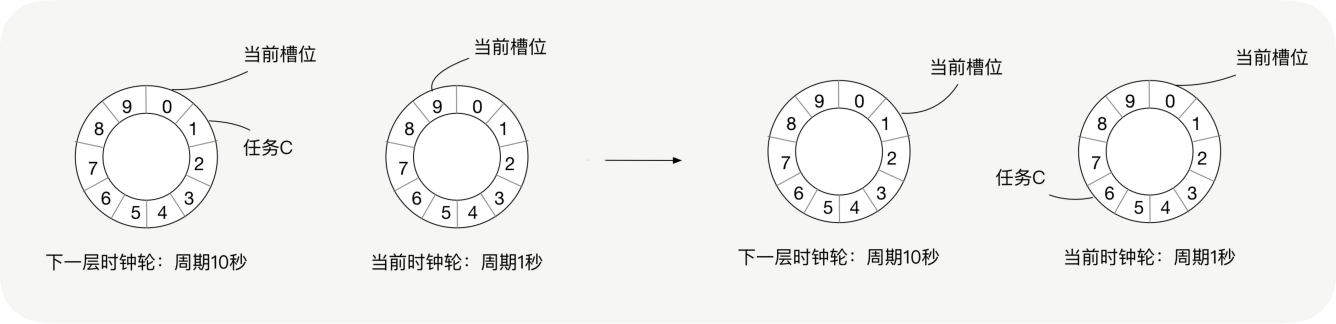
时钟轮示意图

好，现在我们有 3 个任务，分别是任务 A（90 毫秒之后执行）、任务 B（610 毫秒之后执行）与任务 C（1 秒 610 毫秒之后执行），我们将这 3 个任务添加到时钟轮中，任务 A 被放到第 0 槽位，任务 B 被放到第 6 槽位，任务 C 被放到下一层时间轮的第 1 槽位，如下面这张图所示。



时钟轮任务分布示意图

当任务 A 刚被放到时钟轮，就被即刻执行了，因为它被放到了第 0 槽位，而当前时间轮正好跳到第 0 槽位（实际上还没开始跳动，状态为第 0 跳）；600 毫秒之后，时间轮已经进行了 6 跳，当前槽位是第 6 槽位，第 6 槽位所有的任务都被取出执行；1 秒钟之后，当前时钟轮的第 9 跳已经跳完，从新开始了第 0 跳，这时下一层时钟轮从第 0 跳跳到了第 1 跳，将第 1 槽位的任务取出，分布到当前的时钟轮中，这时任务 C 从下一层时钟轮中取出并放到当前时钟轮的第 6 槽位；1 秒 600 毫秒之后，任务 C 被执行。



任务C槽位转换示意图

看完了这个场景，相信你对时钟轮的机制已经有所了解了。在这个例子中，时钟轮的扫描周期仍是 100 毫秒，但是其中的任务并没有被过多的重复扫描，它完美地解决了 CPU 浪费的问题。

这个机制其实不难理解，但实现起来还是很有难度的，其中要注意的问题也很多。具体的代码实现我们这里不展示，这又是另外一个比较大的话题了。有兴趣的话你可以自行查阅下相关源码，动手实现一下。到哪里卡住了，我们可以在留言区交流。

时钟轮在 RPC 中的应用

通过刚才对时钟轮的讲解，相信你可以看出，它就是用来执行定时任务的，可以说在 RPC 框架中只要涉及到定时相关的操作，我们就可以使用时钟轮。

那么 RPC 框架在哪些功能实现中会用到它呢？

刚才我举例讲到的调用端请求超时处理，这里我们就可以应用到时钟轮，我们每发一次请求，都创建一个处理请求超时的定时任务放到时钟轮里，在高并发、高访问量的情况下，时钟轮每次只轮询一个时间槽位中的任务，这样会节省大量的 CPU。

调用端与服务端启动超时也可以应用到时钟轮，以调用端为例，假设我们想要让应用可以快速地部署，例如 1 分钟内启动，如果超过 1 分钟则启动失败。我们可以在调用端启动时创建一个处理启动超时的定时任务，放到时钟轮里。

除此之外，你还能想到 RPC 框架在哪些地方可以应用到时钟轮吗？还有定时心跳。RPC 框架调用端定时向服务端发送心跳，来维护连接状态，我们可以将心跳的逻辑封装为一个心跳任务，放到时钟轮里。

这时你可能会有一个疑问，心跳是要定时重复执行的，而时钟轮中的任务执行一遍就被移除了，对于这种需要重复执行的定时任务我们该如何处理呢？在定时任务的执行逻辑的最后，我们可以重设这个任务的执行时间，把它重新丢回到时钟轮里。

总结

今天我们主要讲解了时钟轮的机制，以及时钟轮在 RPC 框架中的应用。

这个机制很好地解决了定时任务中，因每个任务都创建一个线程，导致的创建过多线程的问题，以及一个线程扫描所有的定时任务，让 CPU 做了很多额外的轮询遍历操作而浪费 CPU 的问题。

时钟轮的实现机制就是模拟现实生活中的时钟，将每个定时任务放到对应的时间槽位上，这样可以减少扫描任务时对其它时间槽位定时任务的额外遍历操作。

在时间轮的使用中，有些问题需要你额外注意：

> 时间槽位的单位时间越短，时间轮触发任务的时间就越精确。例如时间槽位的单位时间是 10 毫秒，那么执行定时任务的时间误差就在 10 毫秒内，如果是 100 毫秒，那么误差就在 100 毫秒内。
>
> 时间轮的槽位越多，那么一个任务被重复扫描的概率就越小，因为只有在多层时钟轮中的任务才会被重复扫描。比如一个时间轮的槽位有 1000 个，一个槽位的单位时间是 10 毫秒，那么下一层时间轮的一个槽位的单位时间就是 10 秒，超过 10 秒的定时任务会被放到下一层时间轮中，也就是只有超过 10 秒的定时任务会被扫描遍历两次，但如果槽位是 10 个，那么超过 100 毫秒的任务，就会被扫描遍历两次。

结合这些特点，我们就可以视具体的业务场景而定，对时钟轮的周期和时间槽数进行设置。

在 RPC 框架中，只要涉及到定时任务，我们都可以应用时钟轮，比较典型的就是调用端的超时处理、调用端与服务端的启动超时以及定时心跳等等。

课后思考

在 RPC 框架中，除了我说过的那几个例子，你还知道有哪些功能的实现可以应用到时钟轮？

欢迎留言和我分享你的答案，也欢迎你把文章分享给你的朋友，邀请他加入学习。我们下节课再见！

更多课程推荐

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　19 | 分布式环境下如何快速定位问题?

下一篇　21 | 流量回放：保障业务技术升级的神器

精选留言 (4)

写留言

嬰
2020-04-06

时钟轮可以实现延时消息的功能，比如让一个任务几分钟之后发送一条消息出去。在比如可以实现订单过期功能，用户下单10分钟没付款，就取消订单，可以通过时钟轮实现。

7

忆水寒
2020-04-06

在java里面和Netty框架里面有这个类，TimeWheel时钟轮模型。

高源

2020-04-06

时钟轮这个我头一次听到，老师如果并发线程比较多，单位时间是不是划分很细啊，但是我有个疑问例如我同时有5个线程几乎之间间隔3到5毫秒，又有3个线程10到100毫秒的，我时钟轮也得调整具体怎么划分的，这么短时间内如何保证时钟轮准确性，老师哪有参考代码学习一下

展开

1 2

qinsi

2020-04-06

时钟轮存取任务的时间复杂度是O(1)，相比之下优先队列的时间复杂度是O(logN)